УДК 004.65

АЛЬ-ЗГУЛЬ МОСАБ БАССАМ

ГИБРИДНЫЕ АЛГОРИТМЫ В СИСТЕМАХ КЭШИРОВАНИЯ ОБЪЕКТОВ

В статье рассматриваются гибридные алгоритмы в системах кэширования объектов. Предлагается метод использования гибридных алгоритмов при построении адаптивных систем кэширования. Описывается новый универсальный метод гибридизации двух и более алгоритмов кэширования, а также новый метод для получения гибридного алгоритма RRFU из алгоритмов LRU и LFU. Приведены результаты сравнительного исследования нового алгоритма RRFU и известного гибридного алгоритма LRFU.

Ключевые слова: *стратегии кэширования, гибридные алгоритмы, RRFU, LRFU против RRFU.*

Введение. Темпы роста вычислительной мощности основных компонентов современных микропроцессорных систем стремительно возрастают. Наиболее быстро развивающейся частью системы является микропроцессор, в то время как эффективность других устройств растет значительно медленнее. Прирост производительности процессоров в настоящее время составляет 50-80% в год, для систем же динамической памяти DRAM такой прирост не превышает 10%. Для увеличения вычислительной мощности системы в целом недостаточно увеличения производительности одного лишь микропроцессора. Эффективным способом использования темпов роста производительности процессоров для компенсации отставания темпов роста скорости доступа к DRAM и внешней памяти является кэширование.

Кэширование данных – это универсальный метод ускорения доступа к данным, основанный на комбинации двух типов памяти, отличающихся временем доступа, объемом и стоимостью хранения данных. Наиболее часто используемая в данный период информация динамически копируется из «медленной, но большой» памяти в «быструю, но маленькую» кэш-память.

Например, если кэширование применяется для уменьшения среднего времени доступа к оперативной памяти, то в качестве кэша выступает быстродействующая статическая память процессора или даже его регистры. А если система ввода-вывода кэширует данные для ускорения доступа к информации, хранящейся на диске, то в этом случае роль кэша исполняют буферы в оперативной памяти, в которых оседают наиболее активно используемые данные.

Кэширование широко используется и в Интернете, в частности при разрешении символьных DNS-имен, в протоколе ARP, в работе браузера. Важную роль системам кэширования отводят и в системах управления базами данных (СУБД) [6]. Принципы организации кэширования на сервере базы данных (БД) во многом сходны с организацией кэшей в операционных системах и микропроцессорах. Резкое удешевление аппаратных средств вычислительной техники привело к тому, что в качестве рабочих станций в

России широко используются достаточно мощные компьютеры. В этих случаях двух- и трехзвенные архитектуры представляют дополнительные возможности организации систем кэширования в СУБД, которые могут иметь большую эффективность из-за возможности использования ресурсов компьютеров пользователей и более полного учета семантики их предметных областей. В таких системах могут быть использованы быстро развивающиеся в настоящее время идеи организации кэширования документов в WEB-системах.

Неотъемлемым свойством кэш-памяти является ее прозрачность для программ и пользователей. Система, реализующая кэширование, не требует никакой внешней информации об интенсивности обращения к данным. Ни пользователи, ни программы не принимают никакого участия в перемещении данных из одного типа памяти в память другого типа. Все это делается автоматически системными средствами.

При каждом обращении к основной памяти (к WEB-странице, блоку, объекту) прежде всего анализируется содержимое кэш-памяти с целью определения нахождения в ней требуемых данных. Далее возможен один из двух вариантов развития событий:

- если данные обнаруживаются в кэш-памяти (cache-hit), они считываются из нее и результат передается источнику запроса;

- если нужные данные отсутствуют в кэш-памяти (cache-miss), они считываются из основной памяти, передаются источнику запроса и одновременно с этим копируются в кэш-память.

Эффективность кэширования зависит от вероятности кэш-попадания. Пусть имеется основное запоминающее устройство со средним временем доступа к данным t_1 и кэш-память, имеющая время доступа t_2, очевидно, что $t_2<t_1$. Пусть t — среднее время доступа к данным в системе с кэш-памятью, а p — вероятность кэш-попадания. По формуле полной вероятности имеем:

$$t=t_1(1-p)+t_2p=(t_2-t_1)p+t_1. \quad (1)$$

Среднее время доступа к данным в системе с кэш-памятью линейно зависит от вероятности кэш-попадания и изменяется от среднего времени доступа в основное запоминающее устройство t_1 при $p = 0$ до среднего времени доступа непосредственно в кэш-память t_2 при $p = 1$. Отсюда видно, что кэширование имеет смысл только при высокой вероятности кэш-попадания [4].

Вероятность обнаружения данных в кэше зависит от разных факторов таких, например, как: объем КЭШа; объем кэшируемой памяти; алгоритм замещения данных в кэше; особенности выполняемой программы; время ее работы; уровень мультипрограммирования и других особенностей вычислительного процесса. Тем не менее, в большинстве реализаций кэш-памяти процент кэш-попаданий оказывается весьма высоким. Это объясняется наличием у данных объективных свойств – пространственной и временной локальности.

Под временной локальностью понимают свойство данных, при наличии которого, в случае, если произошло обращение по некоторому адре-

су, то следующее обращение по тому же адресу с большой вероятностью произойдет в ближайшее время.

Данные обладают пространственной локальностью в том случае, если при возникновении обращения по некоторому адресу с большой вероятностью в ближайшее время произойдет обращение к соседним адресам. Особым видом пространственной локальности является так называемая циклическая локальность, при наличии которой трасса обращения к объектам обладает некоторой цикличностью. Циклические (периодические) трассы могут порождаться при выполнении запросов аналитического и статистического характера [4].

Именно основываясь на свойстве временной локальности, данные, только что считанные из основной памяти, размещают в запоминающем устройстве быстрого доступа, предполагая, что скоро они опять понадобятся.

В начале работы системы, когда кэш-память еще пуста, почти каждый запрос к основной памяти выполняется кэш-промахом, чтением данных из основной памяти, передачей результата источнику запроса и копированием данных в кэш. Затем, по мере заполнения кэша, в полном соответствии со свойством временной локальности, возрастает вероятность обращения к данным, которые уже были использованы на предыдущем этапе работы системы, т.е. к данным, которые содержатся в кэше и могут быть считаны значительно быстрее, чем из основной памяти. Свойство временной локальности в полной мере используется в очень распространенном алгоритме кэширования LRU [7]. Недостатком этого алгоритма является невозможность его противостоять циклической локальности.

Рассмотрим пример обработки циклической последовательности алгоритмом LRU (табл.1), в котором представлена последовательность идентификаторов объектов, вызываемых при выполнении некоторого запроса.

Таблица 1

Пример периодической трассы

Позиция в трассе	1	2	3	4	5	6	7	8	9	10	11	12	13	14	16	17	18
Идентификатор объекта	7	2	1	9	5	3	7	2	1	9	5	3	7	2	9	5	3

Заметим, что у каждого объекта в этой трассе расстояние до его следующего появления постоянно и для простоты принято равным 6. Допустим, что в кэше может поместиться только 5 объектов. Состояние кэша, начиная с позиции 5 трассы (см.табл.1), содержится в табл.2.

Таблица 2

Состояние кэша при выполнении трассы табл.1

Позиции	Состояние кэша					
	1	2	3	4	5	
5	5	9	1	2	7	промах
6	3	5	9	1	2	промах
7	7	3	5	9	1	промах

8	2	7	3	5	9	промах

Как видно из табл.2, LRU на трассе табл.1 обеспечивает 100% промахов. Известно, что для любых модификаций метода LRU можно подобрать трассы, которые дадут также 100% промахов, т.е. работа системы кэширования на таких трассах будет полностью неэффективной [1].

Свойство пространственной локальности также используется для повышения вероятности кэш-попадания: как правило, в кэш-память считывается не один информационный элемент, к которому произошло обращение, а целый блок данных, расположенных в основной памяти в непосредственной близости с данным элементом [2]. Поскольку при выполнении программы очень высока вероятность того, что команды выбираются из памяти последовательно одна за другой из соседних ячеек, то имеет смысл загружать в кэш-память целый фрагмент программы. Аналогично, если программа ведет обработку некоторого массива данных, то ее работу можно ускорить, загрузив в кэш часть или даже весь массив данных. При этом учитывается высокая вероятность того, что значительное число обращений к памяти будет выполняться к адресам массива данных. Значительно сложнее реализовать пространственную локальность в базах данных. Для этого нужны специальные средства [6] и высокая квалификация разработчиков базы, чтобы воспользоваться этими средствами. Кроме того, изменение частоты выполнения различных запросов к базе данных может потребовать изменение структуры первоначального проекта базы в процессе эксплуатации. В сложных информационных системах во время работы большого числа пользователей и в разнородных прикладных задачах происходит постоянное изменение локализации объектов [3]. Производительность системы кэширования в этом случае может быть повышена с помощью комбинирования нескольких алгоритмов. Недостатком таких гибридных схем является повышение их вычислительной сложности, что, при опережающем росте производительности процессоров в настоящее время, не является существенным.

Постановка задачи. Обосновать возможность применения гибридных алгоритмов в адаптивных системах кэширования с целью повышения эффективности таких систем. Разработать простой и эффективный метод гибридизации, с помощью этого метода получить гибридный алгоритм и экспериментально сравнить эффективность полученного алгоритма с известным гибридным алгоритмом.

Метод адаптивного управления системой кэширования. Допустим, что гибридизации подвергаются два алгоритма кэширования и у алгоритма гибридизации имеется управляющий параметр $\lambda \in [0,1]$. Изменение этого параметра приводит к изменению степени влияния алгоритмов, образующих гибрид, на процесс обработки кэш-прерывания. Рассмотрим схему системы управления гибридным алгоритмом кэширования (рис.1).

Для примера на рис.1 показана схема управления системой кэширования СУБД. Работа системы исследуется с помощью программного стенда, в котором поток запросов к базе данных эмулируется с помощью генератора трасс [5]. Кроме генератора трасс, в программный стенд входят: имитатор кэш-системы; монитор трассы; оптимизатор параметра . Монитор трас-

сы запоминает последнюю пройденную часть трассы для дальнейшей оптимизации. Данная часть трассы передаётся оптимизатору параметра , который с помощью алгоритма оптимизации находит такое значение параметра , при котором обеспечивалось бы наименьшее количество промахов к кэш-системе для данного участка. Найденное значение используется кэш-системой во время записи монитором следующего участка трассы. Далее процесс повторяется. Очевидно, что длина участка трассы, передаваемого оптимизатору (обозначим эту длину n), существенно влияет на работу кэш-системы и, следовательно, n может быть вторым параметром системы оптимизации. В рассмотренных примерах принято, что n принимает постоянное значение и равно 3000. Для повышения эффективности в реальной системе монитор трассы и система оптимизации могут быть реализованы на других процессоре и дисковой подсистеме, чем система кэширования и СУБД [8].

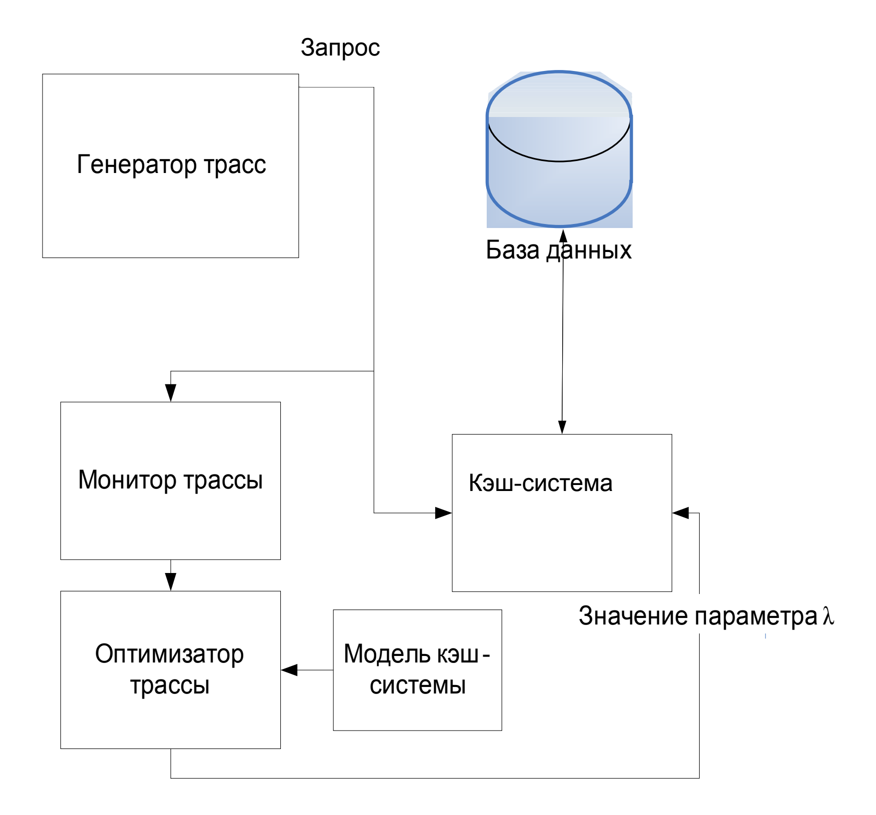


Рис. 1. Схема управления кэш-системой

Методы гибридизации алгоритмов кэширования. Для примера рассмотрим вариант построения гибридного алгоритма Least Recently / Frequently Used (LRFU) на основе комбинирования алгоритмов Least Recently Used (LRU) и Least Frequently Used (LFU) [9].

В алгоритме LRFU для каждого объекта в кэше вычисляется его рейтинг Combined Recency and Frequency (CRF). При вычислении CRF рекурсивно используется так называемая весовая функция вида:

, (2)

где - параметр метода, чаше всего *p*=2; - переменная управления ; - расстояние в трассе между двумя соседними вызовами объекта.

При каждом кэш-прерывании перерассчитывается рейтинг всех объектов кэша и объект с наименьшим рейтингом удаляется из кэша.

В [9] доказывается, что при результаты применения рейтинга CRF полностью совпадают с результатами метода LFU, а при результаты CRF совпадают с LRU.

Эффективность алгоритма LRFU изучалась на примере трассы, сгенерированной с помощью генератора [6]. Вся трасса состоит из трех участков. На первом участке с длиной 30000 обращений генерируются циклические последовательности. На втором участке для генерации потоков запросов используются Марковские цепи [9]. Длина второго участка совпадает с длиной первого. На третьем участке при генерации последовательности обращений вероятность появления любого объекта в трассе постоянна и равна , где *N* – число объектов. Длина последнего участка 15000 обращений.

Результаты испытаний метода LRFU представлены на рис.2, на котором изображены графики изменения процента успешного кэш-попадания для методов LRU, LFU и LRFU на разных этапах реализованной трассы.

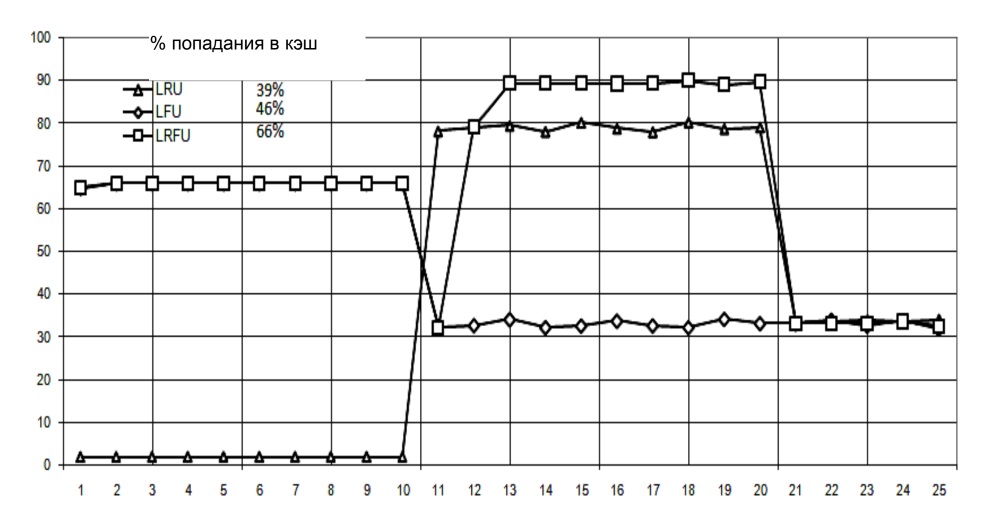


Рис.2. Графики изменения процента попадания в кэш для методов LRU, LFU и LRFU

Как видно из графиков (см.рис.2), эффективность метода LRFU значительно превышает эффективность составляющих его методов LRU и LFU. Недостатком метода LRFU является высокая вычислительная сложность его реализации.

Универсальный метод гибридизации алгоритмов кэширования. Рассмотрим предлагаемый нами метод построения гибридных алгоритмов на примере комбинирования двух известных методов.

Пусть имеется два метода управления системой кэширования, которые в момент t для объектов из кэша дают рейтинг и , где *i*=1,2… *N, N*-размер кэша. Введем управляющий параметр . Пусть в момент времени *t* . По-

лучим с помощью генератора случайных чисел число , при этом, если , для разрешения кэш-прерывания будем использовать рейтинг , а при – рейтинг . При всегда используется только рейтинг , а при только рейтинг . При с вероятностью α будем использовать рейтинг и с вероятностью $(1-\alpha)R_i^2$.

Заметим, что хранение и обработка рейтингов и необязательны. Достаточно, чтобы структуры данных в кэш-памяти поддерживали оба гибридизируемых алгоритма. Таким образом, гибридизации по этому методу могут подвергаться алгоритмы, в которых вообще отсутствует понятие рейтинга, например, алгоритмы случайного выбора объекта для удаления из кэша.

Используем данный метод для получения гибридного алгоритма на базе методов LRU и LFU. Назовем полученный гибридный алгоритм Random Recently/Frequently Used (RRFU). Алгоритм RRFU был реализован на стенде [5], как и алгоритм LRFU, на тех же трассах и с тем же механизмом оптимизации управляющего параметра . Результаты испытаний метода RRFU показаны на рис. 3.

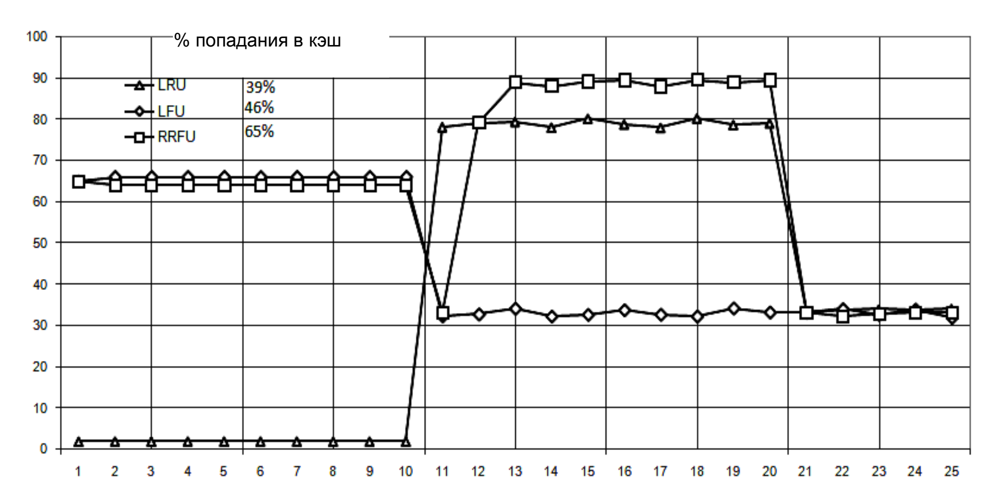


Рис. 3. Графики изменения процента попадания в кэш для методов LRU, LFU и RRFU

Как видно на графиков (см.рис.3), алгоритм RRFU обеспечивает практически ту же эффективность, что и LRFU, однако программная реализация алгоритма RRFU значительно проще, чем у LRFU. Не вызывает никаких сомнений и простота аппаратной реализация метода RRFU.

Выводы. В данной статье предложен метод управления параметрами кэш-системы. Эффективность этого метода показана на примере управления гибридными алгоритмами. Показано, что комбинация алгоритмов кэширования может существенно повысить эффективность системы доступа к информации. Кроме того, предложен простой, эффективный и универсальный метод гибридизации любых алгоритмов кэширования. Простота этого мето-

да позволяет применять его не только в информационных системах и на Web-серверах, но и в операционных системах и процессорах.

Библиографический список

1. Соколинский Л.Б. Стратегия замещения или как освободить место в буфере, проект 00-07-90077 при поддержке Российского фонда фундаментальных исследований, 2002: дис…. д.т.н. – Челябинск, 2003. – С.114-157.

2. Mojtaba Sabeghi, and Mohammad Hossein Yaghmaee 2, Using Fuzzy Logic to Improve Cache Replacement Decisions, March 30, 2006.

3. Dinesh Dasarathan and Santhosh Kulandaiyan, adaptive cache replacement technique, 1998.

4. Олифер В.Г. Сетевые операционные системы. 2-е изд. / В.Г.Олифер, Н.А.Олифер. – СПб: Питер, 2008.

5. Нго Т.Х., Аль-Згуль Б.М. Программный стенд для исследования эффективности алгоритмов кэширования: XXI науч. конф. Т.5. – Саратов, 2008.

6. Вильям Дж., Пейдж. Использование Oracle8/8i. – М., 2000.

7. Таненбаум Э. Современные Операционные Системы. 2-е изд. /Э.Таненбаум. – СПб: Питер, 2002. – С.250-253.

8. Гранков М.В., Нго Тхань Хунг, Аль Згуль Мосаб Басам. Методы разработки тестов на быстродействие информационных систем с использованием цепей Маркова: сб. науч. ст. по проблемам высшей школы. – Новочеркасск: ЮРГТУ, 2006.

9. Donghee Lee, Sam H. Noh, Jongmoo Choi, Sang Lyul Min, Implementation and Performance Evaluation of the LRFU Replacement Policy, Dep. Of Computer Engineering, Seoul National University, Seoul 151-742 Korea.

Материал поступил в редакцию 1.08.08.

MOSAB BASSAM YUSEF AL-ZGOOL

HYBRID ALGORTHIM IN OBJECT CACHE SYSTEMS

In this paper the hybrid algorithms in object caching system is considered. The method using hybrid algorithm in implementation of the adaptive caching systems is offered. The new universal method hybridization of two and more caching algorithms is described. The description of the new hybrid (of the algorithm LRU and LFU) algorithm, namely Random Recently/Frequently Used

(RRFU) is presented. The result of comparative study of the new algorithm RRFU and well-known hybrid algorithm LRFU is showed.

Аль-Згуль Мосаб Бассам Юсеф (р.1981), аспирант кафедры «Программное обеспечение вычислительной техники и автоматизированных систем» ДГТУ по научной специальности 05.13.01 «Системный анализ, управление и обработка информации». Окончил ДГТУ (2005) по специальности «Программное обеспечение вычислительной техники и автоматизированных систем».
Научные интересы: алгоритмы и методы повешения производительности кэш-систем и систем управления распределенными базами данных.
Автор 6 научных работ.